Scaling Scrum Across Modern Enterprises

Implement Scrum and Lean-Agile techniques across complex products, portfolios, and programs in large organizations

Cecil Rupp
Foreword by Manjit Singh (President and CEO, Agilious LLC)

Packt
www.packt.com

Scaling Scrum Across Modern Enterprises

Implement Scrum and Lean-Agile techniques across complex products, portfolios, and programs in large organizations

Cecil Rupp

Packt

BIRMINGHAM—MUMBAI

Scaling Scrum Across Modern Enterprises

Commissioning Editor: Kunal Chaudhari
Acquisition Editor: Alok Dhuri
Senior Editor: Rohit Singh
Content Development Editor: Kinnari Chohan
Technical Editor: Gaurav Gala
Copy Editor: Safis Editing
Project Coordinator: Deeksha Thakkar
Proofreader: Safis Editing
Indexer: Pratik Shirodkar
Production Designer: Shankar Kalbhor

First published: August 2020

Production reference: 1280820

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-83921-647-3

www.packt.com

To my dear wife, Carolyn, who has stood by my side and patiently supported the countless hours I have had to commit to research and writing. Without her support, this work would not have been possible. And, to the loving memory of my parents who instilled in me the values of human decency, hard work, and carrying through a task to its completion.

– Cecil Rupp

Packt.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at packt.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.packt.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Foreword

As an agile practitioner since 2000, I have read hundreds of books on Agile and Lean: books that talk about benefits of Agile and Lean; books that describe one or more Agile frameworks or Lean practices. What I always found to be lacking was a single book that presents the entire landscape of Agile and Lean methods. This book, *Scaling Scrum Across Modern Enterprises*, is the answer!

Cecil "Gary" Rupp is the author of two other recently released books, *SDLC Foundations* and *Tools and Templates*, which are part of his *Building Our Digital World* Series. Gary has done it again! Gary has years of experience of using Agile and Lean methods in information technology consulting, professional services, and executive management roles. *Scaling Scrum Across Modern Enterprises* does an amazing job of presenting the history of Agile methodologies, Lean development, and systems thinking.

As an Enterprise Agility Coach, I have helped customers improve their organizational agility. I have led the evolution of organizations from individuals seeking better ways to deliver products, to small cross-functional teams, to teams of teams with the same purpose in mind.

I have seen first-hand how individuals, team members, and executives struggle with understanding the various Lean frameworks. Further, leaders are looking for answers and a path toward scaling agility. This book provides a comprehensive introduction to and comparison of modern Scrum and Lean-Agile scaling strategies. It demonstrates that modern Scrum and Lean-Agile practices are not just about improving agility in software development, but also about achieving enterprise-wide business agility.

This book provides guidance on how modern scaled Scrum and Lean-Agile approaches help improve business agility across the most challenging organizational structures, product teams, portfolios, and programs. Gary articulates the most important aspects of Agile and Lean that are often overlooked by organizations. This is definitely a must-read for leaders in organizations that are serious about their commitment to scaling agility.

Manjit Singh

President and CEO

Agilious LLC

People.Powered.Agility.

Co-author of the book *The Lean Playbook: Build a Lean Organization Yourself*

Contributors

About the author

Cecil Rupp brings more than 30 years of practitioner and executive-level experience in applying the methods and tools of **information technology (IT)** for software development. His roles span IT professional services, management, business process re-engineering consulting, product management, sales, and marketing.

In addition, Mr. Rupp has directly managed more than 30 enterprise-class IT programs and projects, with the last 15 years focused almost exclusively on supporting large federal and commercial health IT programs. He is also the author of the Building our Digital World (BODW) series of books on software and systems development practices.

From the beginning, I wanted to make sure that I accurately represented the Scrum and Lean-Agile approaches described in this book, and I reached out to the experts to get their input. There are also permissions that must be sought out and granted. So many folks were kind enough to help, and I want to thank all the people who responded back to my queries, read through the sections relevant to their respective disciplines, and who were kind enough to help me with their permissions and to provide their feedback. All of their efforts helped make this a better product.

Especially, I'd like to thank the following for their contributions to reviewing and providing feedback on the content relevant to their disciplines:

Jeff Sutherland – Scrum, Scrum of Scrums, Scrum@Scale

Kurt Bittner and Patricia Kong – The Nexus Framework

Craig Larman and Bas Vodde – Large-Scale Scrum

Scott Ambler – Disciplined Agile

Michelle Stoll – Scaled Agile Framework (SAFe)®

Peter Antman and Henrik Kniberg – Minimum Viable Bureaucracy

About the reviewer

Steve Jablonski is a managing director of technology risk for a large financial services firm. He has over 25 years of technology experience in software development, database administration, project and program management, cybersecurity, governance, and risk management. Steve holds a Master's in Business Administration from the University of Colorado Denver and resides in Littleton, Colorado with his wife, Julie, their two children, and two dogs that enjoy barking and chasing squirrels.

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

2
Scrum Beyond Basics

3
The Scrum Approach

4
Systems Thinking

5

Lean Thinking

6

Lean Practices in Software Development

Section 2: Comparative Review of Industry Scaled Agile Approaches

7

Scrum of Scrums

8

Scrum@Scale

9
The Nexus Framework

10
Large-Scale Scrum (LeSS)

11
Disciplined Agile

12

Essential Scaled Agile Framework® (SAFe®)

13

Full Scaled Agile Framework® (SAFe®)

Section 3: Implementation Strategies

14

Contrasting Scrum/Lean-Agile Scaling Approaches

Assessments

Other Books You May Enjoy

Index

3
The Scrum Approach

The previous chapter introduced the fundamental elements of Scrum, including the importance of executive sponsorship, putting a focus on products, forming Scrum Teams, and identifying Scrum roles and responsibilities, events, and artifacts. This section puts these concepts into play across a Sprint development cycle.

As you read through this section, please refer to *Figure 3.1 – Scrum-based iterative and Incremental development cycle*, which provides a graphical view of the basic flow of work within the Scrum framework. As defined in *The Scrum Guide*, the Scrum events that define a Scrum workflow include **Sprint**, **Sprint Planning**, **Daily Scrum**, **Sprint Review**, and **Sprint Retrospective**. *Figure 3.1* includes additional elements not included in the Scrum Guide, but they provide useful contextual information across the Sprint cycle.

In this chapter, we're going to cover the following main topics:

- Guiding the flow of work in Scrum
- Initiating development work
- How Scrum can break down
- Identifying how Scrum can break down
- Failing implementations of Scrum

Specifically, we are going to put all of the concepts you've learned in the previous chapter together to outline the basic approach to developing a product under the Scrum framework. In this chapter, you will learn how Scrum events enforce the iterative and Incremental workflows of Scrum.

Scrum as a framework

Before we start the discussion on Scrum workflows, we need to understand the ramifications of Scrum being a framework and not an overly prescriptive methodology. In this section, you will learn how to apply the basic Scrum approach to agile as a framework and not an overly prescriptive methodology. By describing Scrum as a *framework*, the implication is that Scrum is a container that provides only minimal guidance on baseline practices, rules, artifacts, and events. The objective of the Scrum philosophy is to keep the essential framework lightweight and relatively simple to understand. Even then, Schwaber and Sutherland note, in *The Scrum Guide™*, that Scrum is still challenging to master.

Since Scrum is a framework, those who implement Scrum are free to include other business and engineering practices that support their approach to software and systems development. The framework concept is critical to understand as the intent of Scrum is to apply agile practices and empirical process control theories to resolve complex adaptive problems across any type of development or operational requirement. However, each organization and their Scrum Teams must choose the life cycle development and delivery practices that best support their needs in the moment.

For example, your development team may use different software tools than other development teams and therefore require a different set of lower-level activities and best practices surrounding the use of those technologies. Likewise, your team may choose to implement test-driven development or model-driven development concepts within the framework of Scrum. Also, your team may implement variants for testing your software, based on the complexity and scale of the code you are developing. More importantly, as your team works together over time and continues to seek constant improvements, you may develop a set of best practices within the Scrum framework that are unique to your team.

As with any development methodology, there is a flow to working within Scrum. The events and artifacts within the Scrum framework support empirical process control through transparency, inspection, and adaptation. But the events and artifacts of Scrum also provide guide rails that constrain work within the iterative and Incremental Sprints of Scrum. The remaining sections of this chapter explain the basic flow of work across each Sprint.

Guiding the flow of work in Scrum

The typical flowchart for Scrum looks quite different from traditional linear-sequential flowcharts of waterfall practices, in part due to the iterative nature of agile-based development practices (see *Figure 3.1 – Scrum-based iterative and Incremental development cycle*). Please refer to the following diagram to see the visual representation of the flow of work within each Scrum Sprint:

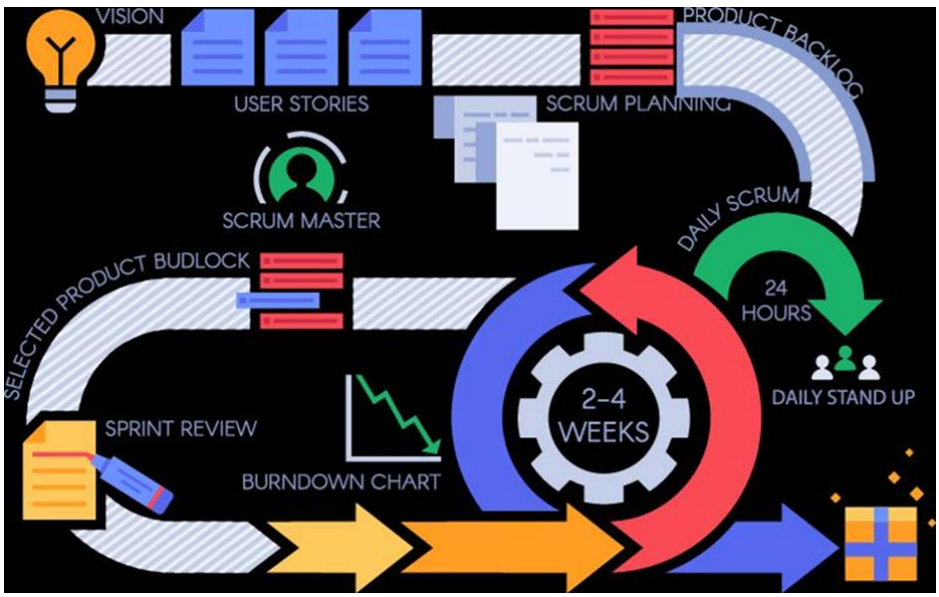

Figure 3.1 - Scrum-based iterative and Incremental development cycle

At the start of a Scrum project, the Product Owner must establish the vision for the product and create the initial product backlog of identified requirements. The vision holds until business or market conditions change sufficiently to warrant a revision. The Product Owner and Scrum Team continuously refine the product backlog to ensure the development activities stay in alignment with the highest value customer priorities.

Once the Scrum project kicks off, the basic flow of work within each Sprint iteration follows this basic pattern:

- Begin a new Sprint.
- Refine the product backlog.
- Determine the Sprint's goal.
- Plan the work.
- Develop the Sprint backlog.

- Conduct Daily Scrum meetings.
- Conduct a Sprint Review.
- Conduct a Sprint Retrospective.

In the remaining sections within this chapter, we will take a deeper dive into each of these Scrum events.

Establishing the product vision

Product development cannot begin until the vision for the product is conceived and articulated. The vision establishes the boundaries of a product. In other words, the vision specifies what's in and what's not in a product.

The vision of the product is not a statement of what it is but what it can be. The product vision refines our understanding of who our customers are and what value we will deliver to them. Moreover, the product vision represents a shared though high-level understanding of our value proposition.

A value proposition is a powerful approach to determining whether or not a new product or service is commercially viable. The information provided in a value proposition typically includes the following:

- Product name and description
- Target market customers
- Challenges or needs addressed
- Capabilities delivered
- Benefits from use
- Competitive advantages

Once the Product Owner establishes the vision for a product, the Sprint iterations can begin.

Implementing iterative and Incremental development cycles

Scrum implements an iterative and Incremental development process that starts with a product concept and vision and then Incrementally adds value through a series of iterative life cycle development workflows. Common with all agile practices, Scrum breaks the product development life cycle into a series of very short and frequent iterations. The objective of each Sprint iteration is to release a new Increment of functionality. Therefore, each Sprint in Scrum represents one iterative and Incremental development cycle.

All Scrum Teams follow the same iterative development cycles. In other words, the scheduling of Sprints across Scrum Teams should not be staggered. They are all contributing collectively to the creation of a potentially shippable product, contributing to the same Sprint Goals and all working toward the same definition of Done for the Sprint.

The Product Owner works with the Scrum Team members to prioritize and select high-value items from within the product backlog that contribute to a specific goal defined for the Sprint. This collaboration to prioritize and select items for upcoming Sprints is called product backlog refinement.

Conducting Product Backlog refinement

Through the process of product backlog refinement, the Product Owner works with the development team members to prioritize the development of features and functions with the highest value. The product backlog refinement process creates and finds the product backlog.

Product owners must determine the highest value features and functions that their product customers and users need. They must also work with the developers to determine the costs associated with developing and delivering new Increments of functionality. Also, there are technical requirements that support the implementation of user requirements. These technical considerations become part of the cost and timing factors associated with developing new product features.

As these factors come together for the highest value features, the Product Owner is in the position to prioritize the items in the product backlog. Following the concepts of the 80/20 rule, the Product Owner and Scrum development team members should not spend much time assessing the work involved to develop lower value features. Some of the lower value items may raise in relative value with the completion of higher value items or through emergent customer needs; but, until they do, the team should not spend much time assessing the work or scoping the requirements.

As part of the product backlog refinement process, the team must analyze each identified requirement to the degree that is necessary to understand the scope of work. There are many approaches to gathering, documenting, and analyzing software and systems requirements. However, within the agile community, the typical approach is to document requirements from the end user's perspective in a story format.

Creating User Stories

The Scrum Guide does not define User Stories as an artifact within Scrum. Kent Beck defined the term and this approach to requirements gathering in his book, *Extreme Programming Explained*.

Nevertheless, Stories are commonly used in Scrum as a natural language format to document customer and end user requirements. Likewise, themes and epics are not Scrum artifacts. However, Stories, themes, and epics are all commonly used within the Scrum framework to characterize and refine items within the product backlog. Collectively, these three classifications provide an efficient approach to documenting and organizing requirements as items within the product backlog.

In the context of Product Backlog refinement, User Stories provide the lowest level of abstraction necessary to define and prioritize work for an upcoming Sprint. During the refinement process, the Scrum Team collects additional information to understand the scope of work that's required. The refinement is complete when the Product Owner and Scrum Team can agree on a definition of Done for each item in the backlog.

Identifying a definition of Done

Another critical element of product backlog refinement is to ensure the team establishes a definition of *Done* for each product backlog item worked within the Sprint. Those who are more familiar with the traditional development model can think of the definition of Done as being analogous to acceptance criteria. In either case, both concepts share common characteristics, such as the following:

- Each requirement should have clear and concise descriptions of what good looks like when correctly implemented.
- The results of the requirement should be testable.
- Everyone on the team needs to understand the requirement.
- Requirements define capabilities that satisfy customer needs and objectives.

The definition of Done is situational to every product backlog item, refined by the Scrum Team members, and ultimately approved by the Product Owner. They must have a common understanding of what good looks like when a feature or function is fully installed and tested in the software or system. Also, there cannot be any bugs in the new code or the integration of the new code with the existing code.

In the last three subsections, you've learned how to refine a product backlog, develop User Stories that further refine the development team's understanding of individual requirements, and specify definitions of Done to help to ensure fulfillment of each backlog requirement. But we also need a method to decide what items within the product backlog should be considered for development within an upcoming Sprint. This process starts with the definition of a Sprint Goal.

Establishing Sprint Goals

Through the Sprint refinement and planning processes, the Product Owner and Scrum Team establish objectives for each Sprint in terms of the implementation of items from the product backlog. The Product Owner and the Scrum Team negotiate objectives and goals for the Sprint. While the Product Owner is accountable for establishing priorities, only the Scrum Team can commit to the work they can accomplish within each Sprint.

Sprint Goals are abstractions that sit above the level of User Stories and work tasks. Let's take a closer look at what I mean by this. If we are building an ATM banking application, we might have a Sprint Goal to build and test a set of features that allow bank withdrawals at an ATM. In this context, we might have two primary User Stories:

- "*As a user of the ATM banking application, I want to see my account balance when I log in so that I know whether I have sufficient funds to withdraw money for my personal needs*."
- "*As the user of the ATM banking application, once I see my available balance, and assuming I have sufficient funds, I want to be able to withdraw as much as $250 from the ATM*."

The Scrum Team defines the work tasks necessary to build these two features within the ATM application. Now the devil is always in the details, and there may be any number of other capabilities that might logically fit and support these two user requirements, such as having the ability to transfer funds between accounts before making a withdrawal and the ability to review pending withdrawals.

The Scrum Team negotiates with the Product Owner to determine which ancillary capabilities are of high value and critical for this release, constrained by the amount of work the Scrum Team can complete during the Sprint. Once they agree, the Scrum Team commits to deliver the negotiated and agreed features and ancillary capabilities within the Sprint.

Much of the work described so far is completed as part of the Sprint Planning event. Also, to the maximum extent possible, most Sprint Planning work is completed within a timeboxed Sprint Planning meeting, as described in the next section.

Conducting Sprint Planning meetings

At the start of the project and the start of each new Sprint, the development team analyzes the highest priority items or stories in the product backlog to identify the deliverable items within an upcoming Increment and how they will go about doing the work. This activity is referred to as Sprint Planning and is the first event scheduled within each Sprint.

The outputs of the Sprint Planning meeting and subsequent breakout sessions include a subset of product backlog items consistent with the Sprint Goals. The Scrum Team refines the agreed definition of Done and creates a list of work tasks and work assignments that are necessary to start building a new Increment of functionality. The sum of the identified work tasks forms the Sprint Backlog, which is necessary before development work can begin in the Sprint.

At the end of each Sprint Planning event, the team should be able to explain the following to both the Product Owner and Scrum Master:

- The Increment of new functionality required to support the Sprint Goal
- The scope of work the Scrum Team expects to accomplish over the Sprint
- A clear explanation of how the team intends to self-organize and allocate their work

At this point, the team is ready to begin working on developing the new Increment.

Initiating development work

All development work in Scrum must fit within timeboxed development iterations that have consistent durations, limited to a period of 1 to 4-week cycles called *Sprints*. The output of a Sprint is an Increment of functionality that meets the definition of Done, is useable without additions or modifications, and is, therefore, a potentially shippable product.

With the definition and refinement of the Sprint Backlog, the development team immediately gets to work starting to build the new Increment of functionality, consistent with the Sprint Goals. Ideally, the teams complete all identified work before the Sprint duration ends, and all completed work complies with the definition of Done.

Recall that Scrum is a framework that serves as a container for other engineering processes. Therefore, test-driven development, continuous integration, and automated testing all logically fit within the Scrum framework and help to ensure the quality of the software.

The Scrum artifacts created and refined within a Sprint include the **product backlog** and **Sprint Backlog** discussed in previous sections of this chapter and **Increments**. The term Increment represents the functionality and value of the product in its current state. An Increment represents the backlog items from the previous Sprint but built on previous Increments. Therefore, while the term Increments represents the current extended functionality of the product, it's also appropriate to think of the term as implying the sum value of the product through the implementation of product backlog items to date.

Within each Sprint are four primary Scrum events: Sprint Planning, Daily Scrum, Sprint Review, and the Sprint Retrospective. We covered the topic of Sprint Planning in the preceding section, so, now, let's take a look, in order, at the Daily Scrums, Sprint Reviews, and Sprint Retrospectives.

Conducting Daily Scrums

Every 24 hours, the Scrum Team meets, ideally at the same and same place, to discuss the progress of the team in completing the work of the Sprint. These Daily Scrum meetings should be short and to the point, usually taking 15 minutes or less. The team does not address any issues in these meetings. Instead, the affected team members and other technical or domain specialists who can help to resolve the issue, take the conversation offline in a separate meeting.

Supporting the pillars of empiricism, the Daily Scrums provide an opportunity for team members to inspect their progress and adapt their work to address any issues. The team can also agree to take on more work if it appears the team will accomplish their Sprint Goals with time to spare.

In the spirit of transparency, executives, customers, end-users, and other stakeholders can join Daily Scrum meetings. However, the Scrum Master must make sure these folks do not interrupt the meeting. The goal of the Daily Scrum is to minimize waste in the form of extended meetings that do not directly support the work of individual team members in their assigned work.

Daily Scrums continue through the duration of the Sprint. At the end of each Sprint, the team conducts a Sprint Review meeting, as discussed in the next section.

Conducting Sprint Reviews

On the last day of the Sprint, the Sprint team meets with designated stakeholders and the Product Owner to review the work of the Sprint. This meeting is called the **Sprint Review**. All work is entirely transparent to the attendees of the Sprint Review. The attendees inspect the work to see whether there are any variances from the original Sprint Goal. The development team members may provide a demo of the new product functionality to prospective users of the software.

The Sprint Review is another form of transparency that allows users and customers to inspect the product to determine whether the new functionality meets their needs. If the new Increment of functionality falls short of expectations or the users have new insights on how the product can better serve their needs, they record that information for further review in upcoming product backlog refinement and Sprint Planning meetings.

During the Sprint Reviews, the Product Owner discusses the work completed in the previous Sprint and the value it provides. They also take the time to discuss current priorities in the product backlog and their future development and release plans. The Product Owner should address performance against the project constraints of schedule, budgets, resources, and quality. Finally, the Product Owner should provide new insights on market conditions, potential business, or customer opportunities and their impact on future development priorities.

The development team reviews their work over the Sprint and explains what went well and any issues they faced and how they addressed those issues. The development team should also provide a demo of the new enhancements to the Sprint Review meeting attendees.

All attendees can participate in the discussions on current priorities, market conditions, new enhancement requests, and future releases. Though no decisions should be made in the Sprint Review, the team should update the product backlog to reflect new backlog items and priorities. The information gathered in the meeting becomes an input into the product backlog refinement discussions and Sprint Planning events.

Before the team meets to plan the next iteration of work, they need to take some time to inspect the work of the previous Sprint. The Sprint Retrospective, discussed in the next subsection, provides a scheduled opportunity for the team to discuss areas where they can adapt to improve their work and remove impediments affecting their ability to complete their work.

Conducting Sprint Retrospectives

Sprint Retrospectives offer an essential opportunity to inspect and adapt the work of the team in light of new information and issues discovered in the previous Sprints. The team's ability to act is highly dependent on the willingness and ability of the team to be transparent in uncovering and discussing their work.

Building strong bonds and trust among team members is critical, as is safety in terms of not making things personal and understanding that discussed information is not ever used to attack other team members. The team is responsible for the work they complete as a group, and they must act and work as a team.

The Sprint Retrospective meeting should occur right after the Sprint Review meeting. The goal of the Sprint Retrospective meeting is to analyze what went well in the previous Sprint and what did not go so well and to discuss ways the team can improve the quality or performance of their work in future Sprints, starting with the very next Sprint. Holding off on scheduling this meeting will make it challenging to implement any desired changes in time to affect the new Sprint positively.

The outputs of Sprint Retrospective meetings are agreements on opportunities to improve the team, and action items to make those improvements. Note that it can take several years for a Scrum Team to fully mature. In the interim, the Sprint Retrospectives help the team to grow and improve in their joint capabilities. As they reach a high level of operation, the Retrospectives help to keep the team from backsliding into old habits or unproductive routines. They also help the team to recognize new opportunities for improvement, including the development of new skills.

The Sprint Retrospective is the last scheduled event in the Sprint cycle. Assuming the Sprint Goals were achieved, with the release of a new Increment of functionality that conforms with the definitions of Done, the output of the Sprint is a potentially shippable product.

Releasing potentially shippable products

At this stage of the Scrum workflow process, the team has developed a new Increment of functionality that conforms with the agreed definition of Done and achieves the Sprint Goals. Products at this stage are potentially shippable. When a development team fails to produce a shippable product, they create an undesirable situation where each new Increment accumulates additional work in progress that the teams must eventually complete.

In other words, if a development iteration has unfinished work, and the Increment does not meet its definition of Done, the team must add the work to a future Sprint. In the meantime, the accumulation of incomplete work makes the product more difficult to work with, and it becomes increasingly more challenging to locate and fix identified bugs. This lack of discipline causes a form of accumulated technical debt that delays delivery of new functionality, slows work down, makes development work more complex, and hides bugs and defects.

So, the objective of Scrum is to have a completed Increment of functionality every Sprint, conforming to the definition of Done and thoroughly tested. The Product Owner, based solely on business reasons, will determine when to release the completed Increments into production. But in the meantime, every Sprint Increment must stand on its own, fully deployable should the Product Owner decides to do so.

That's the end of the basic Scrum workflow. The team moves on to help the Product Owner to refine the product backlog and plan the next Sprint. Now that you understand the basic Scrum workflow, we'll turn our attention to understanding the impact of systems thinking and lean development in the application of both agile and Scrum-based practices.

Identifying how Scrum can break down

Scrum is hard, much harder than it looks from a simple review of the Scrum Guide and memorization of its empirical process control foundations and product-oriented team structures, events, and artifacts. It's even more challenging to scale Scrum across a large product or as an organization-wide implementation.

In this section, you will come to understand that there are innumerable pitfalls that can lead to Scrum implementation failures, both at the product and organizational levels, and how to resolve these issues at the start. Each subsection addresses a particular issue but also provides a discussion on how the organization can avoid or at least minimize the problems.

Lacking executive sponsorship

Executive-level support is a critical success factor for any Scrum implementation. Because Scrum is ultimately about changing the values and the principles that guide the organization, a move to implement Scrum on any scale will run headfirst into impediments created by the organization's culture. Only the most senior executives have the power and authority to remove these impediments.

For example, Scrum requires a movement away from functional departments to product-oriented teams that are self-organizing, self-contained, and autonomous in their efforts to create the highest possible value at the lowest possible cost. The effect is that Scrum eliminates hierarchical organizational structures while changing employee and management positions, roles, and responsibilities. Instead, fully empowered Product Owners supported by their dedicated Scrum Teams replace the bloated bureaucracies of the traditional bureaucratic organizational structures.

Scrum also changes product life cycle development processes to release new Increments of customer valued functionality frequently, forcing the streamlining of all product life cycle development and operations-oriented processes. The streamlining requires more effective communications and collaboration between organizations that previously operated as individual silos. The efforts to streamline all development and operations activities will force the integration of business processes.

Other critical business processes eventually must follow suit. For example, marketing AD campaigns and promotions will have shorter life cycles. Sales organizations must stay current with released product capabilities, features and functions, and the specific customers targeted with each new release. Frequent releases impact product delivery and consulting partner programs, including training and support requirements. When the software is part of a more extensive system, device, or equipment, the changes affect the organization's supply chain partners and associated processes.

So, to recap, Scrum implementations at a project or product development team level have impacts across the organization and its delivery and supply chain partners. The impediments faced by the Scrum Team go beyond their scope of work and authority to address, and, for that reason, executive-level sponsorship is vital. Only a chief or **Line of Business (LOB)** executive has the authority to work through the many organizational issues that will impede the effectiveness of the Scrum Team.

The chief or LOB executive may delegate their authority situationally. But they cannot delegate their authority if they don't know what the issues and impediments are. They must stay informed and engaged in making Scrum work, no matter the scale of the Scrum implementation.

Failing to obtain buy-in

Just because the chief executive is supportive and directing the change to Scrum doesn't mean everyone else in the organization has the same commitments. The lack of organizational buy-in is likely the number one issue the enterprise Scrum implementation team will face. A corporate mandate without proper preparation, communications, early success, and ultimately lack of buy-in will virtually guarantee Scrum implementation failures and delays.

For one thing, organization-wide implementations of Scrum requires major reorganizations, away from hierarchical and functional departments, and to streamlined, product-oriented, and loosely coupled Scrum Teams. Individuals, particularly those in middle management roles, will feel threatened if they don't see a useful role for them and they believe their jobs and compensation are at risk. Yet these are the very people the business needs to buy into the change. If they don't buy in, they will resist the change.

Also, change is scary. However, the odds of achieving early successes improve when the organization stages the roll-out of Scrum through a series of pilot engagements, implemented by the organization's most enthusiastic innovators and early adopters. The odds of success increase, even more, when individuals in other functional groups see an opportunity for them in the change.

There is an adage that says success breeds success. Part of the success comes from hiring people who want to achieve great things. But another critical factor is that most people want to be part of something successful. The successes of the innovators and early adopters generate the enthusiasm required to move the early majority, late majority, and laggards to change and adopt the new Scrum paradigm eventually.

Lacking an agile mindset

Individuals across the organization must develop an agile mindset. Unfortunately, achieving an agile mindset is not all that simple. You cannot merely follow the rules of Scrum to achieve agility. The values and principles of agile must guide decisions made within the Scrum framework and not by the prescriptive rules of Scrum. The Scrum Guide implements few rules and those that exist connect Scrum's roles, events, and artifacts, guiding the relationships and interactions between them.

agile is not a prescriptive methodology with specific rules to follow. Instead, agility is a philosophy expressed as a core set of 4 values and 12 principles. Before an agile framework, such as Scrum, can be implemented, the organization must understand and embrace the core values and principles of agile. Then, and only then, can they begin to figure out how to go about achieving Agility.

agile has a widescale impact on the organization, as identified in the previous two subsections. Ultimately, the culture must change. And since organizational culture is driven by the collective views, objectives, and experiences of its people, culture can only be changed by the people who make up the organization. And that process takes time and work. If a chief executive wants to change the culture, they must generally accomplish the following tasks in roughly this order:

1. Identify the current strengths and weaknesses of the organization's existing culture in terms of values and behaviors.
2. Create and articulate a vision for the future, defining the business strategies, goals, and objectives.
3. Communicate the cultural strengths that can be leveraged but also which values and behaviors need to change and why.
4. Define and communicate the highest value of tactical priorities to meet the organization's strategic goals and objectives.
5. Establish clearly defined goals and metrics to evaluate progress against the tactical plans.
6. Implement the product-oriented teams of Scrum.
7. Establish co-location facilities with both team and individual work areas, plus install development systems and tools and networking and communications infrastructures.
8. Update employee compensation, incentive, and rewards programs to align the progression of skills and team performance with the organization's strategic and tactical goals.
9. Encourage open, honest, and respectful communications in support of Scrum's empirical process control foundations and its pillars of transparency, inspection, and adaptation.
10. Don't just mandate change; create the motivation for change by promoting successes; providing regular feedback, coaching, and mentoring opportunities to Scrum Teams; and recognizing people who demonstrate desired values and behaviors of Agile and Scrum.

Failing to invest

Another vital part of the Scrum implementation preparation is to ensure the organization has the skills, infrastructure, and resources to support the products and Scrum Teams. The implementation of Scrum involves a reinvention of organizational structures, behaviors, and work environments. More substantial investments are required to support enterprise-scale Scrum implementations.

The Product Owners, Scrum Masters, and Scrum Teams will take time to develop their skills in Scrum. I'll discuss training issues in a separate subsection. However, the organization also needs to provide access to individuals who are already trained and skilled in Scrum. Such resources may already exist within the organization. But, in most cases, the organization may need to hire outside consultants to help to guide them through the implementation process.

The organization may choose to create a **Scrum Center of Excellence** (**CoE**) to support, coach, and mentor the newly installed product development teams, Scrum Masters, and Product Owners. An **Executive-level enterprise Scrum Master** (**ESM**), must be installed to work through organizational issues, for example, impediments that require executive-level decision-making and investment authorities. When multiple divisions are involved in the enterprise Scrum implementation, each division should have a dedicated ESM to support their efforts.

The ESMs may establish Scrum Teams solely dedicated to removing organizational-level impediments. Each ESM creates a backlog of prioritized issues that they must address. For example, the ESMs must address gaps in resources, product team alignments, compensation, and incentive plans, knowledge, experience, and infrastructure needs across the organization, both before and during the enterprise deployment of Scrum.

The Scrum Teams also need a physical place to work, ideally in a co-located facility with room to work, conduct breakout sessions, and set up their information radiators. The developers need network access, development and testing computers, and software development and testing tools. These investments advance the effectiveness of the Scrum Team.

Lacking effective communications programs

Organizations that mandate a change to Scrum without preparation are doomed to failure. People need to know why change is required. They want to know what's in it for them. They need to feel they are safe in the change situation, otherwise, they will resist it. Moreover, they need to know what they have to do to be successful in the new environment. None of this can happen overnight.

Start by building a communications plan. The organization likely has the skills in-house to do this, as both marketing staff and project managers have the skills and training to develop and execute effective communications plans. Make sure the communications plan provides a layout of the specific details of why the change is necessary, the timelines, and the expected organizational and personal benefits expected as outcomes of the change.

Make sure employees and managers know who to go to if they have questions or concerns. Provide details on training opportunities and dates. The communications strategy should include information on the staged roll-out priorities and initiation dates as that information becomes available. Also, the communications plan should emphasize early and continuing promotion of implementation successes and the specific accomplishments of individual Scrum Teams.

Failing to educate

It should be evident that organizations that wish to implement Scrum need to train their employees in advance of the implementation and then provide continuing training opportunities as the teams form and mature. But based on my experience, too many companies refuse to make the human and financial resources available to support an effective training program. Even if they do provide access to training resources, how many organizations track and provide incentives to employees who participate in training programs that are relevant to the organization's continued success?

Going back to our discussions on Shuhari, it can take years to master a new subject. The organization might start by providing access to online courses that cover the fundamentals of Scrum. But the organization should also consider bringing in experts to teach Scrum classes and directly address questions unique to the organization's implementation of Scrum. Over time, the organization will produce its experts, and these folks should be available to mentor other Scrum Teams.

Also, the training can't just be about agile and Scrum but should also encompass development practices and skills and the technologies and tools used by the development organization.

The bottom line is learning is an ongoing, never-ending requirement.

Failing implementations of Scrum

In the previous four subsections, you have learned how Scrum implementations fail from lack of executive sponsorships, foundations, agile mindsets, and communications and training programs. In the remainder of this section, you will learn how to resolve the impediments that hinder the successful enterprise or product-level deployments of Scrum.

I've touched on this subject before, but the empirical process control mechanisms of Scrum provide a practical approach to resolving Scrum implementation issues. However, the Scrum Teams cannot resolve most of the issues identified in this section as they don't have the authority to address issues outside their direct product development-related activities.

Therefore, the organization must establish an enterprise-level Scrum CoE or some other type of organizational Scrum implementation resources to resolve issues that require executive-level decisions. These decisions include issues associated with business and organizational alignment, hiring, people management, compensation plans, and investments in infrastructure, tools, training, facilities, and funding.

Adding roles that are not part of Scrum

Scrum Teams only employ three roles, ever. These are the **Product Owner**, the **Scrum Master**, and the **Developers**. The founders of Scrum were very careful not to install structures that would create an overly competitive environment that is not conducive to team building. Any organization that adds additional roles is not truly practicing Scrum. Moreover, another risk from adding roles is organizational bloat and a return to hierarchical and bureaucratic processes.

Focusing on the wrong product backlog items

Given the product-oriented nature of Scrum, there has to be someone responsible for making decisions and ultimately held accountable for the success of the product. To be held accountable, they must have the authority to make decisions on product backlog items and priorities. That is the role of the Product Manager.

There is only one Product Manager assigned to each unique product. In some of the scaled-Scrum approaches introduced in Section 2 of this book, the Product Owner may enlist the help of assistants or Teams of Product Owners working under a Chief Product Owner or Product Manager. For now, let's keep things simple. The Product Owner is the only decision-maker regarding product backlog priorities.

Not even the company's chief executive should override the decisions of the Product Owner. For sure, **Chief Executive Officers** (**CEOs**) and **LOB** executives, customers, end-users, development team members, and other stakeholders will undoubtedly have opinions and seek to influence decisions. Still, there can only be one decision-maker, and that is the person who ultimately has responsibility for the organization's **Return on Investment** (**ROI**) for the product.

Product Owners are the voice of the customer. A successful Product Owner recognizes the customer's interest must come first. If a prospective customer is not happy with a product, they will not buy it. Customers have varying needs and will value different features and functions, price points, performance, and quality uniquely. The challenge is in identifying the right set of product features and functions and priorities to make the product commercially viable.

A product with a large prospective customer base and multiple market segmentation will have a larger pool of requirements options to consider for each Increment. As mentioned in previous sections within this chapter, the Product Owner must work though a multi-dimensional problem that evaluates the size of the market for each identified requirement, the Incremental value to customers for each satisfying requirement, and the cost of producing and delivering each requirement. They must resist every attempt by outside influencers to change priorities that do not fit Scrum's value-based product backlog prioritization model.

Allowing inappropriate priorities

You may think that surely the CEO can change the priorities in the product backlog. However, unless the chief executive is the Product Owner, the answer is no; the CEO cannot change the product backlog items or priorities. The Product Owner cannot allow outside influencers to arbitrarily make decisions on product backlog items and priorities that do not fit the highest-value/lowest-cost prioritization model described previously, no matter who the influencers are.

Product Managers who are weak or ineffective will make bad decisions. For example, they may add low-value items with unsubstantiated priorities within the product backlog. Conversely, they may add items that appear to have high customer value but are not cost-justified given what the target market is willing to pay for them. Or the Product Owner may make an error by purposely choosing to prioritize the development of many low-cost items that have relatively little Incremental value, instead of focusing on the development of higher-value items.

An effective Product Owner understands the scope of knowledge and work that goes into building and sustaining a viable product backlog. When a Product Owner fails, it's likely the hiring chief executive or LOB executive who under-scoped the breadth of skills and cross-functional knowledge required to perform in the role of Product Owner successfully.

For example, the Product Owners must have credible domain knowledge to understand customer issues, capability requirements, and priorities thoroughly. To stay on top of the market trends, the Product Owner needs to spend the bulk of their time meeting with customers, industry analysts, end-users, and stakeholders. Ideally, the Product Owner should seek out opportunities to speak and present at industry forums and become a recognized thought-leader within the industry.

The Product Owner must have the business domain knowledge to assess product pricing strategies accurately, as well as costs, and value. They must have marketing skills and knowledge to understand how to identify target market customers and how best to promote their products. The Product Owner must be a competent business development specialist who can identify new market niches and product opportunities. The Product Owner must also know about sales, including the use of inside and outside sales organizations, and the use of channel partners such as **Value-Added Resellers** (**VARs**), systems integrators, consultants, **Managed Service Providers** (**MSPs**), **Original Equipment Manufacturers** (**OEMs**), and distributors.

Do the Product Owners do all of this work alone? No, of course not. They are the ultimate decision-makers, but they must have access to functional efforts to pull the information together they need to make the right decisions. If the Scrum implementation effort is limited strictly to development activities, the Product Owner will access corporate resources in the functional departments. However, if the organization is implementing Scrum enterprise-wide, the Product Owner can establish functional Scrum Teams to provide support across sales, marketing, partnerships, and other critical product life cycle activities.

In short, the Product Owner must be a capable and knowledgeable *jack of all trades* whose responsibilities encompass the entire scope and breadth of product management. The Product Owner spends a relatively small amount of time directly supporting development-oriented Scrum events, requiring somewhere on the order of only 25% of their time. The Product Owners need time to work in parallel with functional or, more ideally, Scrum-based teams supporting marketing, sales, partnerships, distribution, and other business functions critical to the product's market success.

Directing instead of leading

A common mistake is to place a Scrum Master into a development team based on their technical skills, domain knowledge, or project management experience without making sure they are adequately trained and clearly understand their role. None of those skills are requisite justifications for hiring a Scrum Master. The potential problem is the Scrum Master with such skills may assume authoritarian control over the Scrum Team, which is in opposition to their role in Scrum.

The roles of the Scrum Masters include providing mentoring, coaching, and serving. The Scrum Master provides mentoring and coaching on Scrum practices to the Product Owner, development team members, and any other stakeholders whose views and actions can affect the product development priorities and work. In this role, the Scrum Master monitors decisions and activities, and steps in to provide guidance when the actions are taken or proposed are inconsistent with Scrum.

Moreover, in the role of a servant leader, they help the development team to resolve any impediments that would otherwise distract the team members form to complete the goals of each Sprint. In other words, the authority of the Scrum Master comes not from directives but from their knowledge, their ability to provide guidance, and their desire and ability to serve the team as opposed to leading or directing the team's activities.

Scrum Masters who approach their job as technical leaders make the mistake of trying to exert influence as the arbiter of technology, architecture, design, and development decisions. Such actions fly in the face of both agile values and Scrum practices, where the team as a whole determines the best approach to developing each new Increment of functionality. Scrum Masters who cannot resist influencing technical and development-related decisions should reconsider if they are better suited to working as a development team member.

Scrum Masters coming from a project management role need to understand they are no longer responsible for project planning, scheduling, or prioritization of work. Scrum Masters do not monitor work to control or direct the execution of work. Only the development team, operating as an independent, fully functional, and self-contained unit, can decide how much work they can take on within each Sprint.

If the Scrum Master helps to create the burndown and velocity charts, it is Done in their role to serve the team by allowing the team to stay focused on their value-added tasks. The Scrum Master never develops charts to direct the team's activities. They develop the charts so that the development team, its Product Manager, and other stakeholders can make informed decisions.

When I say the development of burndown and velocity charts are non-valued added work, I don't mean to imply the charts have no value. But the charts are measurements only and do not directly contribute to the development of a new Increment of functionality. On the other hand, the charts are part of the information radiators previously discussed that help to provide transparency and facilitate inspection and adaption processes. So, the charts have value, even though they do not directly contribute to the development of the project.

An easy way to think about value-added versus non-value-added work is to ask yourself whether the activity directly contributes to the construction of the product's Increment. If not, then the work is not value-added from the customer's perspective. The work may be necessary or useful as a vital information radiator item, but the development team cannot let such activities get in the way of developing the Increment. As a Servant Leader, the Scrum Leader steps in to help the team.

Likewise, rather than use the information from the burndown and velocity charts to micromanage and direct the team's progress, the Scrum Master provides the information freely and without judgment and as an aid to the team's decision-making process. The goal of a Servant Leader is to help the team to obtain the information so they can assess the progress of their work and their capacity as a team.

Having risks and issues are the norm and not the exception when developing large and complex software and IT-aided systems. Risk management is the reason Scrum implements the concepts of empirical process control. It's impossible to know or plan for every contingency that affects a development project. In some cases, the development team responds directly to address any issues that arise. However, the Scrum Master should work through any issues not directly related to the act of designing and building the Increment. They may schedule fact-finding meetings or need to resolve other issues that affect development team member participation.

Performing non-value-added activities

The fastest way to kill an Increment is to allow the development team to get sidetracked on non-value-added activities or work. Non-value-added work is any activity that does not directly contribute to the development of the current Increment of functionality as required to achieve the Sprint Goal. Activities that can take away the development team's focus includes the attendance of non-relevant meetings, working on development tasks or other activities not related to the current Increment, and working on issues best resolved by the Scrum Master. Over-engineering a product beyond what is required to achieve the Sprint Goal is another example.

The timeboxed events of Scrum (for example, Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective) provide maximum transparency while minimizing time spent in non-value-added meetings. The Scrum Master helps to ensure the teams stick to the scope of each event and within the time boundaries. When an additional discussion is required to resolve a development issue, the impacted development team members should take the meeting offline and allow the other team members and meeting participants to get back to their work.

In an ideal situation, the development team has a typical area in which they work as a team. The co-location of team members facilitates the display and use of information radiators, paired programming, and breakout sessions to discuss requirements and architecture, design, and coding issues. The developers need access to networks, computing systems, and software development tools. I also personally believe people need time and space to think without distraction and to decompress. Having cubicles or individual desks within or adjacent to the shared meeting room provides that individual space.

Scrum development teams are self-organizing. That means they must have the authority to evaluate the work necessary to fulfill a Sprint's goal and make individual assignments based on priorities, time, skills, and interests. Scrum development teams are self-contained. That means each team must have the skills necessary to complete all work. Since Scrum development teams are limited in size, the individuals must value learning new skills and developing competency in multiple skills. New product requirements and new technologies and tools will expand over time the skills needed to develop the team's assigned products. As a result, employee compensation and incentive plans must support the multi-learning objectives of Scrum.

With all of these constraints in mind, the development teams cannot afford team members who are specialists. Still, there will be times when the need for a new skill is immediate, and the team may not have the expertise needed to develop a new Increment of functionality. In those cases, the team must have access to specialists on a time-limited basis. Those specialists may come from an internal group or hire through outside contractors. However, in the longer term, assuming the new skills become an ongoing requirement, the development team should build the skills in house.

Allowing team burnout

A critical issue that comes up time and again is burnout of Scrum development team members. This issue happens when the Scrum Masters, Product Owners, chief, or LOB executives exert too much influence on individual Scrum goals and over commit the developers. Only the development team can make commitments on the scope of work they can take on during each timeboxed Increment and plan the work tasks necessary to achieve the Sprint Goal.

Executives, Product Owners, and Scrum Masters who do not recognize the development team must have the final say on Sprint Goals and work tasks make two mistakes:

- They have not understood that value trumps functionality.
- They have not understood the importance of transparency.

In the first case, Product Owners who focus on making value a priority understand that releasing something customers want and will pay for is more important than waiting to implement a product with the most features and functions. The 80/20 rule is a prime consideration, as we've learned previously that ~20% of a product's features provide approximately 80% of the product's value. The other 80% of the identified prospective backlog items are expensive and time consuming to deliver and offer little value in return to the customer.

In the second case, transparency is critical as it provides timely and accurate information for accurate decision making. Transparency enables trust, which is why Scrum's three pillars of empirical process control, *transparency*, *inspection*, and *adaptation,* are so important. The organization's executives and Product Owners may question whether the development team is putting their full effort toward meeting their target release dates. Such concerns are reasonable and open for discussion. However, interference with a team's decisions is not. Only the Scrum Team can adequately access the scope of work they can take on within an Increment. But the Scrum Team also has a responsibility to provide sufficient and accurate information to have informed conversations.

For example, if the Product Owner or executives have concerns about the team's *velocity*, they can have a conversation to see whether there are external impediments that are limiting the scope of work completed within each Increment. It also might make sense to evaluate the economic feasibility of adding additional Scrum Teams to help to work through the product backlog.

Failing to provide full transparency

Scrum development teams, when properly implemented, can determine the amount of work they can complete withing an upcoming Sprint. They must have the authority to make decisions on workloads. But they must also show accountability through the visibility of their velocity charts and by meeting their commitments.

Besides the Scrum events, velocity and burndown charts and other useful metrics provide evidence of the team's ability to both judge and complete the work they have committed to complete within each Sprint and planned for future releases. Over time, the development team builds a profile of their capabilities by estimating the work they can complete in each Sprint, often expressed as story points, and then tracking their performance against their estimates. The team charts this data across each completed Sprint. The measure of their performance, expressed as story points per Sprint, is called velocity, and the charts showing velocity over time are called velocity charts.

Burndown and velocity charts are not the only items that provide transparency on the team's activities and capabilities. The team can employ any number of handwritten, drawn, printed, or electronic displays of their work and their decisions. For example, they may have the User Stories written on 3" by 5" index cards and posted on a Scrum Board to show those that are in a queue, work in progress, tested, and Done. They may have architecture and design drawings displayed on whiteboards or flip charts. They may also draw out the screen displays, application reports, and screen navigation features. Test scripts and test results should be displayed.

There is no limit to what can be displayed so long as the information is useful and relevant. Such information, when publicly displayed for all to see, is referred to as an *information radiator*.

Continuing development beyond economic value

We can think about this issue in another way; how many features in a mature word processing application do you use, let alone across the entire suite of products? As an author, there may be certain features I use more than you might in a word processor. There may be other features you use that I do not. Other users will have different feature sets they prefer. The question is how many features have the team implemented that do not have sufficient economic value to justify the effort?

Figure 3.2 – Word processor application needs

One of the primary roles of the Product Owner is to look at the intersection of our needs and other market opportunities, to determine the sweet spot for maximizing value at the lowest production delivery costs. See *Figure 3.2* on *'Word Processor Application Needs'* as an example. The **Sweet Spot** identified in the graphic offers the maximal economic return to the organization's investments in the product as it includes the subset of features desired by all types of customers.

That's not to say the sales opportunities within the author, market, or your collection of needs might not economically justify further investments. The Product Owner needs to gauge whether features should all go into a generalized product or whether it might be better to offer niche variants of the product. A generalized product costs less to promote, sustain, and sell. However, niche products may avoid turning off customers who believe the full-featured product has become too complicated. Niche products may also support a higher price and a higher ROI.

The Scrum Teams stay together for as long as the addition of the new features and functions continue to add value sufficient to justify continued investments. Of course, the product's accumulated costs continue to increase with added development activities. But that does not matter so long as the revenues from new product sales and from existing customers for maintenance, support, and upgrades offset the ongoing costs of continued product development activities.

Failing to support market segment opportunities

Some products have a large and diverse customer base. The Product Owner, working with the product's marketing staff, segments their market opportunities based on groupings of common characteristics, such as customer demographics, interests, needs, and locations. The Product Owners follow the same rules of prioritizing identified backlog items with the highest-value and lowest-cost.

Take a quick look at the graphical example of *Figure 3.3* – Market Segmentation Priorities – Intersecting, which is a generalized view of the graphic presented in *Figure 3.2*. You can see there is an area of overlap where the customers' *needs* span all three market segments, for example, the **Sweet Spot!** From Increment to Increment—this is the area where the Product Owner knows they can maximize sales opportunities:

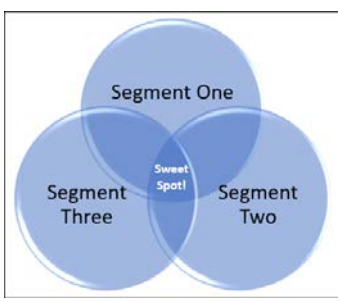

Figure 3.3 – Market Segmentation Priorities – Intersecting

At some point in time, the developers may complete the implementation of all features within the intersecting needs of the three marketing segments. Now the market segments must carry their weight to justify further development investments.

The reason for pointing this out is that markets are seldom this easy to segment. For example, *Figure 3.4* displays a situation where target market segment two does not intersect the customer needs of either target market segments one and two. There is a sweet spot among the needs of the first two market segments, and that may be a logical place to start development.

However, what do you do if the customers in segment three will pay more for a product than the customers in market segments one and two? Be careful here. The easy answer is to assume we'll build the product for segment three customers. However, the needs may be so unique and challenging that the development costs outstrip the additional revenues.

In the meantime, perhaps segment two customers will pay more for additional features, beyond the sweet spot, that are relatively simple and inexpensive to implement. Now the Product Owner should consider making those segment two features a higher priority, along with those identified in the sweet spot:

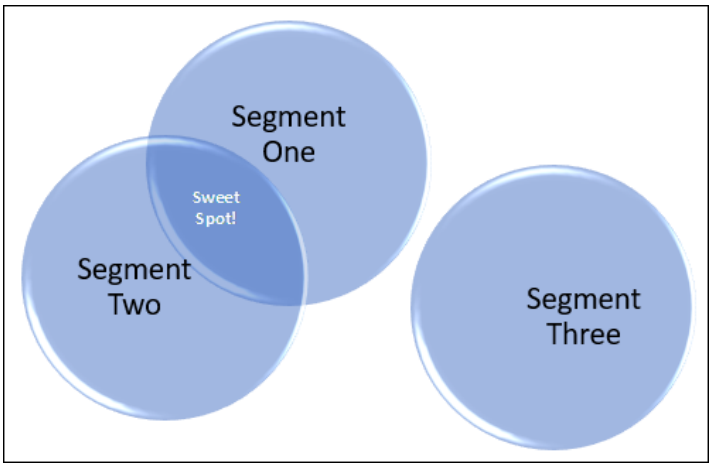

Figure 3.4 - Market Segmentation Priorities – Non-Intersecting

The bottom line is that, across Increments, the product backlog may include items that address the needs of one or more market segments. There still is only one product backlog. Also, the Product Owner still makes the priorities based on the highest-value/lowest-cost prioritization model.

Regardless of the situation, when producing a new release for a product with multiple market segments and niches, your marketing and sales campaigns must be in sync for each new release. Otherwise, your company may not achieve the sales goals that justified the investments. In other words, the Product Owner cannot solely focus on development priorities; they must also make sure the rest of the product marketing, sales, delivery, and support functions are operating in sync.

Pushing deliveries beyond capacities

Companies exist because there is a profit incentive to create things that customers want. The same paradigm holds for government agencies and non-profits. Rather than profit, legislative mandates and goodwill drive government agencies and non-profits to create products and services their customers want. Motivation is useful in that it drives our economies and citizen support systems.

But unbridled motivation can also destroy a company by releasing products and services that are not ready for delivery. When we are not honest, disaster follows—usually in missed delivery dates, cost overruns, and reduced profitability.

Here, again, the answer is transparency. Product Owners need to determine and communicate the identified product backlog items with the highest customer-centric value. The Scrum teams need to make visible their capacities to deliver. Executives need to communicate to investors and their customers the organization's capacities and plans to deliver.

Putting undue pressure on the development teams will not fix the problems of misinformed expectations. Moreover, putting more pressure on the development organization is likely to backfire, creating stress and long hours that hinder the team's performance. The developers need to work at a sustainable pace or they will burn out and mentally and emotionally check out.

The development team's primary focus must continuously remain on only providing the highest value product features with the lowest cost, across each development iteration. But that statement also assumes you have a legitimate market opportunity and capacity to deliver within a competitive timeline. If your competitors beat you to the market and your organization does not have a compelling and unique value proposition, then, most likely, your company shouldn't be making this product or service anyway.

Failing to work as a team

This subsection discusses a catch-all area of behaviors that can hinder the success of a Scrum Team. For example, a dominant team member may seek to lead instead of jointly collaborating on critical decisions. The Scrum Master needs to get involved and mentor the team member on more effective ways to work with their team members.

Some team members may not pull their weight at work. For example, a team member may be late to Scrum events or may not adequately participate and engage in the development work. They may also have a limited skill set and may not learn new skills that would otherwise help the team to more efficiently self-organize, be self-sufficient, and evenly allocate work across each Increment.

The team can positively address these concerns with the individual during their Sprint Retrospective meetings, though the discussions need to remain respectful to retain the integrity of the team. If the member is defensive or non-responsive, the Scrum Master needs to get involved, listen to everyone's concerns, and see whether there is a resolution that works for the individual and the team.

Failing to evolve the product Incrementally

Sometimes, a new product concept is enormous in scope and complexity, making it difficult to immediately assess the features, functions, priorities, and architecture and design requirements. And, in some cases, it may be difficult to refine the vision without some experimentation. But how is the team supposed to proceed in those cases?

After all, without a solid vision and specific product goals and objectives, the development team can't know how to get started. If they start on development, they can't know what they need to deliver. Finally, without a complete vision, the team won't know whether they are off track working on items that have little or no value.

Still, we have to start on something. Though it may be tempting, it's not a great idea to put a new product out to market too quickly. It's much better to build a series of prototypes until the functionality reaches a stage that the product has enough value to attract customers and end-users.

I'll admit, this approach is not textbook Scrum. But as digital remove systems continue to merge into increasingly complex products, we need to manage our risks. It takes time to figure out what our customers want, and releasing a product before it's ready will do more harm than good. It's better to set expectations correctly upfront with customers, stakeholders, and investors.

Keeping the development focus on continually delivering only the highest-value Increments allows the product to mature gracefully. It also provides that fastest path to deliver a viable release.

The value-cost development priority model does not change. The Product Owner still defines a prioritized list of requirements within the product backlog. The development team works through the product backlog as expeditiously as possible, but not to the point of exhaustion and burn out. It's the CEO's job to manage shareholder or customer expectations. As mentioned in the Scrum Team burnout section, productivity will go down if the work pace is not sustainable, and your best people will leave.

By definition, prototypes are potentially disposable products. The reason for this is the developers, the Product Owner, and the paying customer or executive sponsor must be able to walk away from an early architecture or design that cannot meet the business goals that justified the investments. Modern evolutionary architecture concepts help to address these risks by allowing the architecture to emerge in lock-step with the product.

Prototypes also allow the customer to provide early guidance from customers and end-users. The team should find out early on what the customers and end-users do not like or want in the product so that they can cut their losses and remain focused on developing the features and functions the customers do want. The development team works through multiple development iterations until the baseline product can justify a release of the product to customers. The Product Owner decides with input from targeted customers when the product is ready for release.

Summary

At the beginning of this chapter, you learned some of the history and basic concepts behind Scrum. Later, you were introduced to the roles and responsibilities, events, and artifacts associated with Scrum. We learned that modified Scrum is no longer Scrum and why. We also learned that enterprise Scrum is hard to implement as it requires a change in the culture of the organization. Moreover, the changes will remove layers of middle management, and those people must have new opportunities within the organizational deployment of Scrum, or they will resist all efforts to make the deployment successful.

This chapter presented the basic workflow associated with the iterative and Incremental development cycles of Scrum, which are called Sprints. In this section, you learned the use and purpose of Scrum roles, events, and artifacts across each Sprint.

In the next chapter, you are going to learn about *systems thinking*. Systems thinking is not a software development methodology. Instead, it is a way of thinking about complex systems to understand how the collective parts work as a whole to accomplish some purpose or function. However, it's important that you understand the fundamentals of systems thinking as many of the scaled Scrum and Lean-agile practices you'll learn about in *Section 2* of this book employ these concepts.

Questions

1. Why is Scrum described as a framework?
2. How does the traditional development model most differ from the Scrum model?
3. Who has the final say on the scope of work that a Scrum Team can complete within a Sprint?
4. Why does the Product Owner have the final say on the items and priorities established within the product backlog?
5. What is the purpose of the Daily Scrums?
6. What is the purpose of the Sprint Reviews?
7. What is the purpose of the Sprint Retrospectives?
8. What are some of the issues that can cause a Scrum Team to fail?
9. What is the potential problem with hiring a Scrum Master based solely on their technical skills, domain knowledge, or project management experience?
10. What is the primary issue with continuing to develop a product beyond its economic value?